Customer Churn Prediction

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline
```

Load the data

```
df = pd.read_csv("Customer_Churn.csv")
df.head(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mult
0	7590-VHVEG	Female	0	Yes	No	1	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	
4	9237-HQITU	Female	0	No	No	2	Yes	

5 rows × 21 columns

First of all, drop customerID column as it is of no use

```
df.drop('customerID',axis='columns',inplace=True)
```

```
df.dtypes
```

```
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

TotalCharges should be float but it is an object.

```
df.TotalCharges.values
```

```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

convert it to numbers

```
pd.to_numeric(df.TotalCharges)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string " "

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
```

2 frames

```
/usr/local/lib/python3.10/dist-packages/pandas/_libs/lib.pyx in
pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string " " at position 488
```

Next steps: Explain error

some values seems to be not numbers but blank string

```
pd.to_numeric(df.TotalCharges,errors='coerce').isnull()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
7038    False
7039    False
7040    False
7041    False
7042    False
Name: TotalCharges, Length: 7043, dtype: bool
```

```
df[pd.to_numeric(df.TotalCharges,errors='coerce').isnull()]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
488	Female	0	Yes	Yes	0	No	No phone service
753	Male	0	No	Yes	0	Yes	No
936	Female	0	Yes	Yes	0	Yes	No
1082	Male	0	Yes	Yes	0	Yes	Yes
1340	Female	0	Yes	Yes	0	No	No phone service
3331	Male	0	Yes	Yes	0	Yes	No
3826	Male	0	Yes	Yes	0	Yes	Yes
4380	Female	0	Yes	Yes	0	Yes	No
5218	Male	0	Yes	Yes	0	Yes	No
6670	Female	0	Yes	Yes	0	Yes	Yes
6754	Male	0	No	Yes	0	Yes	Yes

```
df.shape
```

```
(7043, 20)
```

```
df.iloc[488].TotalCharges
```

```
' '
```

```
df[df.TotalCharges!=' '].shape
```

```
(7032, 20)
```

Remove rows with space in TotalCharges

```
df1 = df[df.TotalCharges!=' ']
df1.shape
```

```
(7032, 20)
```

```
df1.dtypes
```

```
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
<ipython-input-18-b67e0c3d31a6>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

```
df1.TotalCharges.values
```

```
array([  29.85, 1889.5 ,  108.15, ...,  346.45,  306.6 , 6844.5 ])
```

```
df1[df1.Churn=='No']
```

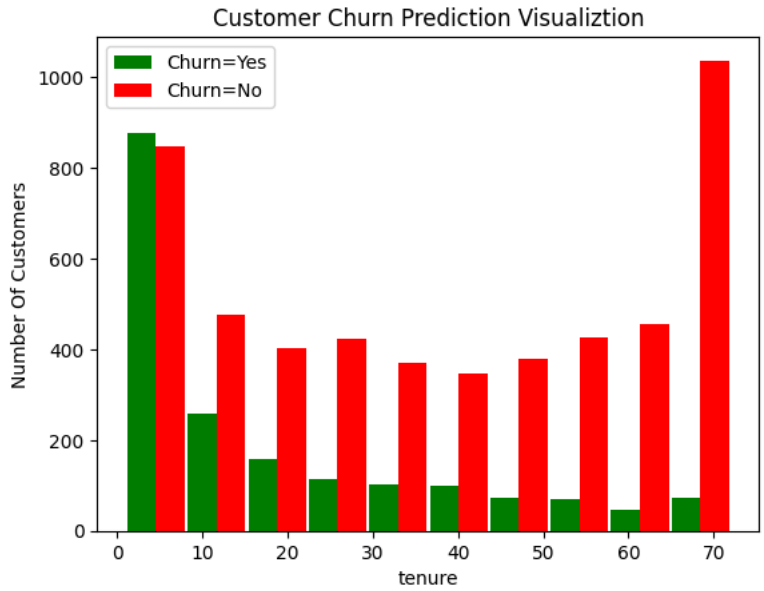
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	Female	0	Yes	No	1	No	No phone service
1	Male	0	No	No	34	Yes	No
3	Male	0	No	No	45	No	No phone service
6	Male	0	No	Yes	22	Yes	Yes
7	Female	0	No	No	10	No	No phone service
...	...	...	...	...	...	...	..
7037	Female	0	No	No	72	Yes	No
7038	Male	0	Yes	Yes	24	Yes	Yes
7039	Female	0	Yes	Yes	72	Yes	Yes
7040	Female	0	Yes	Yes	11	No	No phone service
7042	Male	0	No	No	66	Yes	No

5163 rows × 20 columns

Data Visualization

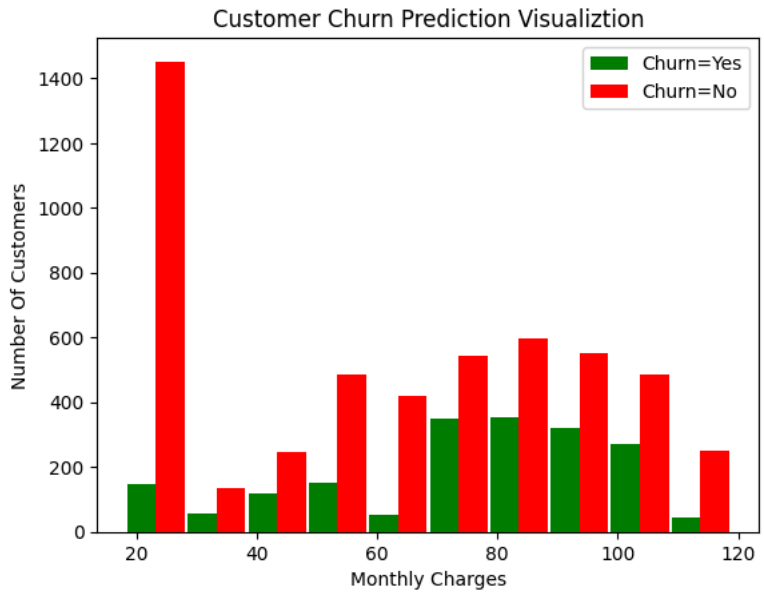
```
tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure

plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")

# blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
# blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

plt.hist([tenure_churn_yes, tenure_churn_no], rwidth=0.95, color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7eb699b04f10>
```




```
mc_churn_no = df1[df1.Churn=='No'].MonthlyCharges
mc_churn_yes = df1[df1.Churn=='Yes'].MonthlyCharges

plt.xlabel("Monthly Charges")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")

# blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
# blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

plt.hist([mc_churn_yes, mc_churn_no], rwidth=0.95, color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7eb69786e8c0>
```



Many of the columns are yes, no etc. Let's print unique values in object columns to see data values

```
def print_unique_col_values(df):
       for column in df:
            if df[column].dtypes=='object':
                print(f'{column}: {df[column].unique()}')
```

```
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No phone service' 'No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['Yes' 'No' 'No internet service']
DeviceProtection: ['No' 'Yes' 'No internet service']
TechSupport: ['No' 'Yes' 'No internet service']
StreamingTV: ['No' 'Yes' 'No internet service']
StreamingMovies: ['No' 'Yes' 'No internet service']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

Some of the columns have no internet service or no phone service, that can be replaced with a simple No

```
df1.replace('No internet service','No',inplace=True)
df1.replace('No phone service','No',inplace=True)
```

```
<ipython-input-25-104b877f3854>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1.replace('No internet service','No',inplace=True)
<ipython-input-25-104b877f3854>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1.replace('No phone service','No',inplace=True)
```

```
print_unique_col_values(df1)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes']
OnlineBackup: ['Yes' 'No']
DeviceProtection: ['No' 'Yes']
TechSupport: ['No' 'Yes']
StreamingTV: ['No' 'Yes']
StreamingMovies: ['No' 'Yes']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

Convert Yes and No to 1 or 0

```
yes_no_columns = ['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','OnlineBackup',
                  'DeviceProtection','TechSupport','StreamingTV','StreamingMovies','PaperlessBilling','Churn']
for col in yes_no_columns:
    df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
<ipython-input-27-34dfac0bf179>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col].replace({'Yes': 1,'No': 0},inplace=True)
```

```
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

```
gender: ['Female' 'Male']
SeniorCitizen: [0 1]
Partner: [1 0]
```

```
Dependents: [0 1]
tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService: [0 1]
MultipleLines: [0 1]
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: [1 0]
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges: [  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
Churn: [0 1]
```

```
df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
<ipython-input-29-ba153b6b6960>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1['gender'].replace({'Female':1,'Male':0},inplace=True)
```

```
df1.gender.unique()
```

```
array([1, 0])
```

One hot encoding for categorical columns

```
df2 = pd.get_dummies(data=df1, columns=['InternetService','Contract','PaymentMethod'])
df2.columns
```

```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

```
df2.sample(5)
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceProtectio
6703	1	0	0	0	2	0	0	0	0	
5251	1	1	0	0	5	1	0	0	0	
5376	0	0	0	0	18	1	1	0	0	
5587	1	1	0	0	46	1	0	0	1	
6713	0	1	0	0	68	1	1	0	1	

5 rows × 27 columns

```
df2.dtypes
```

```
gender                                     int64
SeniorCitizen                              int64
Partner                                    int64
Dependents                                 int64
tenure                                     int64
PhoneService                               int64
MultipleLines                              int64
OnlineSecurity                             int64
OnlineBackup                               int64
DeviceProtection                           int64
TechSupport                                int64
StreamingTV                                int64
StreamingMovies                            int64
```

```
PaperlessBilling                           int64
MonthlyCharges                           float64
TotalCharges                             float64
Churn                                      int64
InternetService_DSL                        uint8
InternetService_Fiber optic                uint8
InternetService_No                         uint8
Contract_Month-to-month                    uint8
Contract_One year                          uint8
Contract_Two year                          uint8
PaymentMethod_Bank transfer (automatic)    uint8
PaymentMethod_Credit card (automatic)      uint8
PaymentMethod_Electronic check             uint8
PaymentMethod_Mailed check                 uint8
dtype: object
```

```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

```
for col in df2:
    print(f'{col}: {df2[col].unique()}')
```

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [0.         0.46478873 0.01408451 0.61971831 0.09859155 0.29577465
 0.12676056 0.38028169 0.85915493 0.16901408 0.21126761 0.8028169
 0.67605634 0.33802817 0.95774648 0.71830986 0.98591549 0.28169014
 0.15492958 0.4084507  0.64788732 1.         0.22535211 0.36619718
 0.05633803 0.63380282 0.14084507 0.97183099 0.87323944 0.5915493
 0.1971831  0.83098592 0.23943662 0.91549296 0.11267606 0.02816901
 0.42253521 0.69014085 0.88732394 0.77464789 0.08450704 0.57746479
 0.47887324 0.66197183 0.3943662  0.90140845 0.52112676 0.94366197
 0.43661972 0.76056338 0.50704225 0.49295775 0.56338028 0.07042254
 0.04225352 0.45070423 0.92957746 0.30985915 0.78873239 0.84507042
 0.18309859 0.26760563 0.73239437 0.54929577 0.81690141 0.32394366
 0.6056338  0.25352113 0.74647887 0.70422535 0.35211268 0.53521127]
PhoneService: [0 1]
MultipleLines: [0 1]
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
PaperlessBilling: [1 0]
MonthlyCharges: [0.11542289 0.38507463 0.35422886 ... 0.44626866 0.25820896 0.60149254]
TotalCharges: [0.0012751  0.21586661 0.01031041 ... 0.03780868 0.03321025 0.78764136]
Churn: [0 1]
InternetService_DSL: [1 0]
InternetService_Fiber optic: [0 1]
InternetService_No: [0 1]
Contract_Month-to-month: [1 0]
Contract_One year: [0 1]
Contract_Two year: [0 1]
PaymentMethod_Bank transfer (automatic): [0 1]
PaymentMethod_Credit card (automatic): [0 1]
PaymentMethod_Electronic check: [1 0]
PaymentMethod_Mailed check: [0 1]
```

Train test split

```
X = df2.drop('Churn',axis='columns')
y = df2['Churn']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=5)
```

```
X_train.shape
```

```
(5625, 26)
```

```
X_test.shape
```

```
(1407, 26)
```

```
X_train[:10]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceProtect
5664	1	1	0	0	0.126761	1	0	0	0	
101	1	0	1	1	0.000000	1	0	0	0	
2621	0	0	1	0	0.985915	1	0	0	1	
392	1	1	0	0	0.014085	1	0	0	0	
1327	0	0	1	0	0.816901	1	1	0	0	
3607	1	0	0	0	0.169014	1	0	1	0	
2773	0	0	1	0	0.323944	0	0	0	0	
1936	1	0	1	0	0.704225	1	0	1	1	
5387	0	0	0	0	0.042254	0	0	0	0	
4331	0	0	0	0	0.985915	1	1	0	0	

10 rows × 26 columns

```
len(X_train.columns)
```

```
26
```

Double-click (or enter) to edit

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
model_params = {
    'svm': {
        'model': SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10]
        }
    },
}
```

```
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(X_test, y_test)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

	model	best_score	best_params
0	svm	0.790298	{'C': 1, 'kernel': 'linear'}
1	random_forest	0.762621	{'n_estimators': 10}
2	logistic_regression	0.783900	{'C': 5}

Next steps: Generate code with df View recommended plots

```
model = SVC(gamma='auto', C=1.0, kernel='linear')
```

```
model.fit(X_train,y_train)
```

```
SVC
SVC(gamma='auto', kernel='linear')
```

```
model.score(X_test, y_test)
```

```
0.7860696517412935
```

```
yp = model.predict(X_test)
yp[:5]
```

```
array([0, 0, 0, 1, 1])
```

```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

```
y_pred[:10]
```

```
[0, 0, 0, 1, 1, 1, 0, 1, 0, 0]
```

```
y_test[:10]
```

```
2660    0
744     0
5579    1
64      1
3287    1
816     1
2670    0
5920    0
1023    0
6087    0
Name: Churn, dtype: int64
```

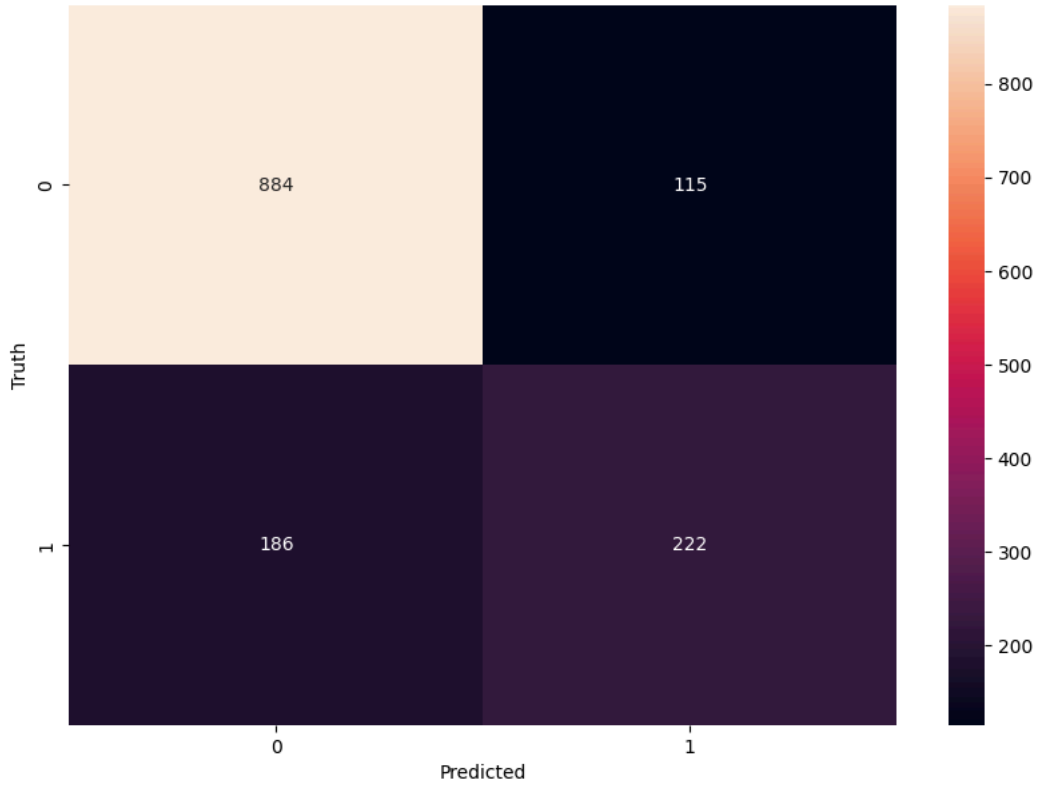
```
from sklearn.metrics import confusion_matrix , classification_report
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.88      0.85       999
           1       0.66      0.54      0.60       408

    accuracy                           0.79      1407
   macro avg       0.74      0.71      0.73      1407
weighted avg       0.78      0.79      0.78      1407
```

```
import seaborn as sn
from sklearn import metrics
cm = metrics.confusion_matrix(y_true=y_test,y_pred=y_pred)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(95.72222222222221, 0.5, 'Truth')
```




```
y_test.shape
```

```
(1407,)
```

```
import pickle
with open('churn_model.pickle','wb') as file:
    pickle.dump(model,file)
```

```
import json
columns = {
    'data_columns' : [col.lower() for col in X_test.columns]
}
with open("columns2.json","w") as f:
    f.write(json.dumps(columns))
```

Start coding or generate with AI.